Efficient Trace Generation for Rare-Event Analysis in Chemical Reaction Networks

Bryant Israelsen[1 [0000−0002−9537−2645]], Landon Taylor[1 [0000−0002−4071−3625]], and Zhen Zhang[1 [0000−0002−8269−9489]]

Utah State University, Logan, UT, USA
{bryant.israelsen, landon.jeffrey.taylor, zhen.zhang}@usu.edu

Abstract. Rare-events are known to potentially cause pathological behavior in biochemical reaction systems. It is important to understand the cause. However, rare events are challenging to analyze due to their extremely low observability. This paper presents a fully automated approach that rapidly generates a large number of execution traces guaranteed to reach user-specified rare-event states for Chemical Reaction Network (CRN) models. It is enabled by a unique combination of a multi-layered and service-oriented CRN formal modeling approach, a dependency graph method to aid the shortest rare-event trace generation, and randomized compositional testing. The resulting prototype tool shows marked improvement over stochastic simulation and probabilistic model checking and it offers insights into a CRN.

Keywords: Compositional testing, rare-events, dependency graph

1 Introduction

As a formalism for modeling chemical kinetics, *Chemical Reaction Networks* (CRNs) are widely used for modeling biochemical reaction systems [6], genetic regulatory networks [29], and molecular programming [38]. Many biochemical systems are intrinsically stochastic, including processes in gene and protein expressions. Essentially, their constituent chemical reactions are often simultaneously enabled to occur in parallel with different probabilities. Moreover, their noisy operating environment can introduce unexpected behavior. Rare events in these systems are often of significant relevance, because they represent extreme infrequent occurrence of undesirable behavior that may lead to pathological effects. Therefore, obtaining provable reliability guarantees is a must for CRNs. *Probabilistic model checking* (PMC) can provide such quantitative guarantees and allows *in silico* analysis for detecting and quantifying rare errors. However, PMC approaches are challenged by the need for enumerating a model's large or even infinite state space to gather a sufficient number of rare-event traces in order to provide accurate probability verification. This task is typically computationally intractable. Further, for probabilistic analysis, it is often necessary to gather many traces that reach the rare-event states of interest. Generating only a small number of them is often insufficient to give an accurate estimate.

This paper presents a fully automated approach that rapidly generates a large number of execution traces guaranteed to satisfy a user-specified rare-event property for a CRN model. These traces are used to compute a lower probability bound for the rare-event property. We first propose a novel multi-layered, service-oriented, and modular CRN modeling approach using the IVy modeling language [27]. It offers flexibility in customizing both the reaction execution frequency and the length of traces. We then propose a dependency graph method to guide the shortest trace generation with unique finite prefixes through compositional testing. These traces are guaranteed to reach the specified rare event and are collected to compute the rare-event's lower probability bound. The dependency graph also effectively proves unreachability of a given rare event, leading to considerable savings in performance. We implemented these methods in a prototype tool, *Random Assume Guarantee Testing Induced Model Executions for Reachability* (RAGTIMER), and found preliminary results to be promising. The proposed rare-event trace enumeration technique can potentially be integrated with many formal and semi-formal rare-event analysis methods and the generated traces can provide detailed debugging information for understanding reachability of rare-events. We believe that this unique combination of the presented methods has not been proposed elsewhere, and is an effective alternative to rare-event simulation approaches for biochemical reaction networks.

2 Preliminaries

Chemical Reaction Networks (CRNs). Under the *Stochastic Chemical Kinetic* (SCK) model assumption, the time-evolution of a CRN is governed by the Chemical Master Equation. Formally, a CRN is a tuple $\mathcal{M}$ composed of n chemical species $\mathcal{X} = \{\mathcal{X}_1, \ldots, \mathcal{X}_n\}$, m reactions $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_m\}$, and an initial state representing each species' molecule count $s_0 : \mathcal{X}^n \to \mathbb{Z}_{\geqslant 0}$. Given a reaction $\mathcal{R}_i$, denote $\mathsf{Reactant}_i \subseteq \mathcal{X}$ as the reactant set and $\mathsf{Product}_i \subseteq \mathcal{X}$ as the product set in $\mathcal{R}_i$. A reaction $\mathcal{R}_i = \langle \alpha_i, v_i \rangle$ includes a *propensity function* $\alpha_i : \mathbb{Z}^n_{\geqslant 0} \to \mathbb{R}^+$ corresponding to the probability (including 0) for it to occur in a state and the *state change vector* $v_i \in \mathbb{Z}^n$ corresponding to the update in molecule count for each species due to reaction $\mathcal{R}_i$. Under the SCK assumption, each reaction $\mathcal{R}_i$ occurs nearly instantaneously, which practically limits both v_i to the values of $0, \pm 1, \pm 2$, and the size of $\mathsf{Reactant}_i$ to be less than three [29].

CRN Semantics. A CRN under the SCK assumption induces a *Continuous-time Markov Chain* (CTMC), where state change due to a reaction occurs in discrete amounts and the probability of state change is dependent on real-valued time. A CTMC model $\mathcal{C}$ is a tuple $\mathcal{C} = \langle \mathbf{S}, s_0, \mathbf{R}, \mathbf{L} \rangle$ where $\mathbf{S}$ is a finite state set (i.e., *state space*); $s_0 \in \mathbf{S}$ is the sole initial state; $\mathbf{R} : \mathbf{S} \times \mathbf{S} \to \mathbb{R}_{\geqslant 0}$ is the transition rate matrix; and $\mathbf{L} : \mathbf{S} \to 2^{AP}$ is a state labeling function with atomic propositions AP. Transition rate $\mathbf{R}(s, s')$ from state s to s' is determined by the propensity of $\mathcal{R}_i$, assuming $\mathcal{R}_i$ is the sole reaction causing this state change. The propensity is the number of possible combinations of reactant molecules:

$\alpha_i(s) = k_i \prod_{\mathcal{X}_j \in \mathsf{Reactant}_i}(s[j])$, where $\mathcal{R}_i$'s *reaction rate constant* is $k_i \in \mathbb{R}^+$. A reaction $\mathcal{R}_i$ is *enabled* to occur in state s if its corresponding propensity function evaluates to a positive value, i.e., $\alpha_i(s) > 0$. Often, multiple reactions are enabled to occur in state s and the corresponding probability for $\mathcal{R}_i$ is $p(s, s') = \frac{\mathbf{R}(s,s')}{\mathbf{E}(s)}$, where the *exit rate* $\mathbf{E}(s) = \sum_{s' \in post(s)} \mathbf{R}(s, s')$ sums up all enabled reaction rates in s. A CTMC model has a non-zero probability of staying in a state and the probability of leaving a state s within time interval $[0, t]$ is $1 - e^{\mathbf{E}(s)\cdot t}$, where t is a non-negative real-valued quantity representing real time.

Time-bounded Reachability Property and Target States. We focus on computing the following non-nested time-bounded transient reachability probability specified in *Continuous Stochastic Logic* (CSL) [3,20]: $\mathsf{P}_{=?}(\Diamond^{[0,T]}\ \Psi)$. It queries the probability of reaching the rare-event Ψ-states within T time units. Let condition Ψ be $\mathcal{X}_\Psi = C_\Psi$, where $C_\Psi \in \mathbb{Z}_{\geqslant 0}$ and $s_0(\mathcal{X}_\Psi) \neq C_\Psi$. That is, a target is an equality condition for exactly one species and it is not initialized to the target value. A state is a *target state* s_Ψ if Ψ evaluates to `true` in s_Ψ, i.e., $s_\Psi \models \Psi$. This work aims at efficiently providing the guaranteed lower probability bound, i.e., $\mathsf{P}_{min}(\Diamond^{[0,T]}\ \Psi)$, where Ψ is $\mathcal{X}_\Psi = C_\Psi$. Note that the user is not required to provide an upper bound for each species, which could induce an infinite-state CTMC. However, the method presented in this paper only generates finite traces where the last state is a target state. Therefore, the resulting CTMC constructed from these traces have a finite state space.

Compositional Testing. A CRN model $\mathcal{M}$ consists of interacting chemical reactions where each executes atomically. $\mathcal{M}$ is a *closed* system, meaning that it does not require any external input, because reactants required by one reaction are provided by other reactions in the same model. These features naturally allow for compositional testing. As detailed in Section 6, a CRN model can be represented as a composition of two interacting processes p_1 and p_2, denoted as $p_1 \| p_2$, following the circular assume-guarantee reasoning rules shown below:

$$\frac{\langle\alpha\rangle\ \ p_1\ \ \langle\gamma\rangle \qquad \langle\gamma\rangle\ \ p_2\ \ \langle\alpha\rangle}{\langle\text{true}\rangle\ \ p_1 \| p_2\ \ \langle\alpha \wedge \gamma\rangle} \quad (1) \qquad \frac{\langle\alpha\rangle\ \ p_1\ \ \langle\alpha\rangle \qquad \langle\alpha\rangle\ \ p_2\ \ \langle\alpha\rangle}{\langle\text{true}\rangle\ \ p_1 \| p_2\ \ \langle\alpha\rangle} \quad (2)$$

The triple $\langle\alpha\rangle p_1 \langle\gamma\rangle$ in Rule (1) can be understood as follows. From the start of process execution, up to step $k-1$, if p_1 satisfies its environment *assumption* α, in the form of Boolean-valued constraints on p_1's input, then the allowed input and output behavior of p_1 determines the *guarantee* γ in the current execution step k. Similarly, process p_2 in the triple $\langle\gamma\rangle p_2 \langle\alpha\rangle$ guarantees α at the present if γ holds in the past. This interpretation avoids the circular definition of this rule by requiring that each process in the composition *only* relies on the correctness of inputs it received in the *past*, but not those to be received in the future, in order for its output to satisfy their respective guarantees. Therefore, as long as neither assumption fails first, neither guarantee can fail first in the composition, and hence $p_1 \| p_2 \models \alpha \wedge \gamma$. For the triple $\langle\alpha\rangle p_1 \langle\alpha\rangle$ in Rule (2), it is interpreted

as that p_1 does not cause the global property α to fail. This rule states that if neither process in the composition causes α to fail first, then α always holds.

Predicated on Rules (1) and (2), *compositional testing* [11,25,28] is a semi-formal technique that empirically checks satisfiability of the guarantee γ for each triple $\langle\alpha\rangle p_i\langle\gamma\rangle$ in the composition by *sampling* inputs from those satisfying the assumption α. Generation of inputs typically involves randomization. For Rule (2), testing of the triple $\langle\alpha\rangle p_i\langle\alpha\rangle$ includes both generating only inputs satisfying α and verifying that the outputs of p_i do not fail α. Each process $p_i = (I_i, O_i)$ consists of an input action set I_i and an output action set O_i. Denote a_i^m as action a_m *defined* in process p_i. a_i^m is an *input action* for p_i if a_i^m is called by another process p_j $(j \neq i)$, but is an *output action* for p_j. Execution of a_i^m modifies a non-empty set of local variables in p_i, and each variable $v \in V_i$ is bounded by a range R_v. Two processes p_1 and p_2 are compatible for composition if $O_1 \cap O_2 = \emptyset$, and their composition is $p_1 \| p_2 = (I, O)$, where $I = (I_1 \cup I_2) \backslash (O_1 \cup O_2)$ and $O = (O_1 \cup O_2)$.

3 Related Work

Rare-event properties are challenging to analyze due to their extremely low observability. *Statistical model checking* (SMC) techniques (e.g., [30,43,24]) have integrated rare-event methods, including *importance sampling* [16,17,14] and *importance splitting* [18,36,42]. Importance sampling biases simulation by weighting the rare-event probability to increase its observability and then compensates for the loss to yield the unbiased probability [24]. Importance splitting reformulates a rare-event probability as a product of less-rare level-conditional probabilities [15]. For analyzing rare-events in biochemical systems, the *weighted Stochastic Simulation Algorithm* (wSSA) [19] relies on a user-defined biasing scheme to favor reactions leading to observing the rare-event, but is limited by the user's insight in selecting the proper biasing scheme. Extensions of wSSA (e.g., [14,33]) have substantially improved its computational efficiency. Recent algorithms (e.g., [35]) can characterize rare events in terms of system parameters. As an alternative, *weighted ensemble* [44,2] has been configured to sample rare events in CRNs [9,45]. Importance splitting divides a model's state space into contiguous levels ordered in the increasing likelihood of reaching the rare event [23,41,24]. The crux of it is the (possibly manually constructed) importance function, which rewards a simulation trace by spawning multiple copies if it crosses a level closer to the rare event, but terminates it otherwise. In [4], the authors presented an automated compositional importance function derivation technique based on the model structure and the rare-event property. Recently, the extended RESTART with *prolonged retrials* [39,40] importance technique was re-implemented in the SMC engine `modes` [4,5] in the MODEST TOOLSET [12].

Advantages of the proposed approach over rare-event simulation. First, it is *fully automated* and neither requires expert knowledge of nor poses modeling limitations on the CRN model. Secondly, it is potentially less computationally intensive

as it neither requires rare-event biasing computation nor wastes any simulation traces not able to reach the rare event. Lastly, simulation-based approaches provide an *estimate* of the actual rare-event probability, whereas the proposed method provides a guaranteed lower probability bound.

4 Motivating Example

The motivating example is the *modified yeast polarization* model [7], a CRN consisting of seven species reacting through eight reactions:

$$\mathcal{R}_1:\ \emptyset \xrightarrow{0.0038} \mathrm{R}, \quad \mathcal{R}_2:\ \mathrm{R} \xrightarrow{4.00\times10^{-4}} \emptyset, \quad \mathcal{R}_3:\ \mathrm{L}+\mathrm{R} \xrightarrow{0.042} \mathrm{RL}+\mathrm{L},$$
$$\mathcal{R}_4:\ \mathrm{RL} \xrightarrow{0.010} \mathrm{R}, \quad \mathcal{R}_5:\ \mathrm{RL}+\mathrm{G} \xrightarrow{0.011} \mathrm{G_a}+\mathrm{G_{bg}}, \quad \mathcal{R}_6:\ \mathrm{G_a} \xrightarrow{0.100} \mathrm{G_d},$$
$$\mathcal{R}_7:\ \mathrm{G_d}+\mathrm{G_{bg}} \xrightarrow{1.05\times10^{3}} \mathrm{G}, \quad \mathcal{R}_8:\ \emptyset \xrightarrow{3.21} \mathrm{RL}.$$

All reaction propensities are in molecules per second. The initial molecule count for the following species vector $(R, L, RL, G, G_a, G_{bg}, G_d)$ forms the initial state $s_0 = [50, 2, 0, 50, 0, 0, 0]$. This system was modified from the pheromone induced G-protein cycle in Saccharomyces cerevisia [10] with a constant population of ligand (L = 2) preventing it from reaching equilibrium [34]. The rare event is a measure of an unreasonably rapid build-up of G_{bg}. Thus, the property of interest is the probability that the molecule count for G_{bg} reaching 50 within 20 seconds: $\mathsf{P}_{=?}(\Diamond^{[0,20]}\, G_{bg} = 50)$. The high concurrency nature of this model is evidenced by $\mathcal{R}_1$ and $\mathcal{R}_8$ each being independent of all other reactions and enabled in all states. Additionally, it takes at least 100 reaction executions to reach a target state. These features can easily overwhelm state expansion methods performed by probabilistic model checking tools as discussed in Section 10.

5 Method Overview

Figure 1 shows a logical flow of the proposed novel approach for the RAGTIMER tool, where the steps in blue symbolize looping behavior. It first reads in a user-specified CRN $\mathcal{M}$ and a rare-event property of interest. A dependency graph is then generated for the given CRN model and target property reachability is determined, as described in Section 7. The dependency graph information is then used to automatically generate the service-oriented layered IVy model (Section 6), which is used with compositional testing to generate the desirable shortest traces with unique prefixes as described in Sections 7 and 8. Stochastic simulation is then performed on each trace to obtain its execution probability, and a summary of these results is returned to the user (Section 9).

RAGTIMER significantly differs from statistical model checking techniques that *estimate* the rare-event probability by biasing events leading to the rare-event during stochastic simulation (e.g., importance sampling) or by incrementally selecting and spawning simulation traces with higher likelihood of reaching the rare event (e.g., importance splitting) [24]. Instead, RAGTIMER produces numerous (shortest) traces *proven* to terminate in a rare-event state, essentially

performing a partial state space exploration of the model, and then computes the cumulative probability of each trace.

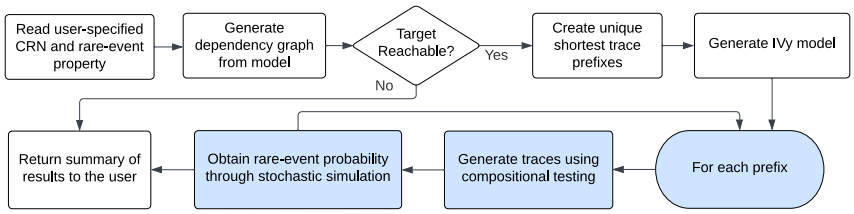


Fig. 1: RAGTIMER Flowchart

6 Layered and Service-Oriented CRN Model Generation

Conventionally, a CRN is modeled as a set of concurrently executing guarded commands, each presenting a constituent chemical reaction, such as those produced by the SBML-to-PRISM converter [21,1]. The modeling approach in this work presents a fresh perspective by considering a CRN as a layered set of service objects that maintain all of its constituent chemical reactions:

1. Layer 0 includes the following objects: `enabled_checker` to evaluate a reaction's readiness to occur, `selector` to select an enabled transition to execute, `updater` to update species as the result of a reaction, `inspector` to monitor reaction behavior, and `goal` to check reachability of the desired target.
2. Layer 1 includes a top-level object `protocol` to manage the execution of all constituent reactions of the CRN by calling services at the lower layer.

In a CRN model $\mathcal{M} = \langle \mathcal{X}, \mathcal{R}, s_0 \rangle$, every reaction $\mathcal{R}_i \in \mathcal{R}$ is modeled as an action `update_Ri` in `protocol`. Actions in `enabled_checker` check whether a given reaction has sufficient reactant(s) to occur at each state. Actions in the `selector` object can be configured to determine the frequency of executing enabled reactions. The `updater` object has actions to increment or decrement a given species according to the state change vector for $\mathcal{R}_i$. The action in the `goal` object monitors whether an execution sequence of reactions has reached a state where the goal is achieved. Note that the layered and service-oriented IVy model presented in this section omits reaction rates and hence is probability-abstract due to IVy's lack of support in floating point operations needed for computing probabilities. Acquisition of rare-event probability is described in Section 9.

The layered modeling approach naturally facilitates modularity and rare-event trace generation using compositional testing, which is a feature provided by the IVy verification tool [28,27]. Consider `protocol` as process p_1 as shown in Rule (2). A mirror process for `protocol` and all layer-zero objects together

```
object protocol = {
 before update_R3 {
  assert enabled_checker.
    is_enabled_R3(r, l)}
 action update_R3 = {
  if selector.execute_R3 {
   call inspector.
    check_guard_R3(r, l, rl);
   r := updater.decr(r);
   l := updater.decr(l);
   rl := updater.incr(rl);
   l := updater.incr(l) }}}
```

```
object enabled_checker = {
 action is_enabled_R3(r_1:
    updater.num, l_1:updater.
    num) returns(y:bool) = {
   if r_1 >= 1 & l_1 >= 1 { y
    := true }
   else { y := false }}}
object inspector = {
 before check_guard_R3 {
  assert r_1 >= 1 & l_1 >= 1}
 action check_guard_R3(r_1:
    updater.num, l_1:updater.
    num, rl_1:updater.num)}
```

Fig. 2: IVy model snippet showing `protocol`'s `update_R3` action calling other actions in lower-layer objects to execute reaction $\mathcal{R}_3$.

form the environment process p_2. The *mirror* process is one in which no actions are defined and its only purpose is to nondeterministically call the *exported* actions defined in p_1. An exported `idling` action is also defined in p_1, which becomes the *only* available action after the goal is achieved. It models that $\mathcal{M}$ becomes idle after a target state has been reached for the first time. This enables us to effectively curtail the model execution trace after it reaches the goal. All layer-zero objects also constituting p_2 each define actions that can only be called by p_1, effectively contributing to output actions for p_1. Rare-event trace generation is achieved by compositionally testing the triple $\langle\alpha\rangle p_1 \langle\alpha\rangle$ in isolation. Isolation of p_1 (i.e., `protocol`) is with respect to its environment p_2. For correctness, checking of p_1's outputs is predicated only on p_2's assumptions. Both assumption and guarantee formulas are declared as assertions in the IVy model. When p_1 is checked in isolation, assertions taking place before p_1 action calls become assumptions, and those following p_1 action calls become guarantees. IVy's compositional testing tool then generates randomized inputs satisfying the assumptions α, while checking that the guarantees α are not violated.

We use reaction $\mathcal{R}_3$ at s_0 in the motivating example as an illustration. As Figure 2 shows, before `update_R3` can happen, $\mathcal{M}$ first checks its precondition (expressed as an assertion) by calling action `is_enabled_R3` defined in `enabled_checker`. It checks whether $\mathcal{R}_3$ has sufficient reactants to occur at the current state. If so, it is enabled and `update_R3` calls `selector.execute_R3`, which determines whether to execute $\mathcal{R}_3$. Only when it is selected can the call be made to action `inspector.check_guard_R3`, whose `before` monitor checks the sufficient precondition again for $\mathcal{R}_3$. The assertion in `is_enabled_R3` of `enabled_checker` is converted to an assumption, but the one in `check_guard_R3` of `inspector` becomes a guarantee for action `update_R3`. Note that these two assertions are functionally equivalent. For example, they both check that the following specification α, `r_1 >= 1 & l_1 >= 1`, holds during compositional test-

ing. When an assumption for this action fails to hold, model execution skips this action. However, violation of a guarantee will halt model execution and report a failure. This guarantee is checked by `inspector.check_guard_R3`, rather than the earlier call to action `enabled_checker.is_enabled_R3`. This is because the purpose of `is_enabled_R3` is to determine whether $\mathcal{R}_3$ is enabled to occur during model execution, but it may not be selected even if it is enabled during testing. Therefore, having a guarantee in this action similar to that of `inspector.check_guard_R3` actually leads to incorrect behavioral modeling, since the execution should not stop when $\mathcal{R}_3$ is merely disabled. However, it is necessary to guarantee sufficient reactants in `inspector.check_guard_R3`, because in order to reach this point, $\mathcal{R}_3$ must have already been selected to occur. Failure of this guarantee stops the execution, as it reveals a flaw in the model. If $\mathcal{R}_3$ successfully executes, `update_R3` updates its reactants and products by calling actions in `updater`. Lastly, the `protocol`'s environment process arbitrarily chooses another exported action to execute and this procedure repeats until the goal is achieved, after which, only the `idling` action is allowed to execute.

A *run* for $p_1 \| p_2$ is a finite sequence of action calls made by both p_1 and p_2. A *valid run* r is a finite sequence of action calls made by both p_1 and p_2 ending with $\texttt{idling}_1$ or $\texttt{goal}_2$. Recall that the subscript indicates the process in which an action is defined. Define reaction $\mathcal{R}_i$ as a finite sequence of actions with the prefix $\texttt{is\_enabled\_Ri}_2$, $\texttt{update\_Ri}_1$, $\texttt{execute\_Ri}_2$, $\texttt{check\_guard\_Ri}_2$, and followed by a continuous sequence of either $\texttt{incr}_2$ or $\texttt{decr}_2$. A (valid) *trace* σ is a finite sequence of reactions obtained by removing actions in r that do not belong to a reaction. An example of a valid run is: $\texttt{is\_enabled\_R5}_2$, $\texttt{is\_enabled\_R3}_2$, $\texttt{update\_R3}_1$, $\texttt{execute\_R3}_2$, $\texttt{check\_guard\_R3}_2$, $\texttt{decr}_2$, $\texttt{decr}_2$, $\texttt{incr}_2$, $\texttt{incr}_2$, ..., $\texttt{goal}_2$. In this run, p_2 tries to execute `update_R5`, but fails because $\mathcal{R}_5$ is disabled. It then succeeds in executing $\mathcal{R}_3$, making it the first reaction in the extracted trace. Note that our model preserves the atomicity of a single reaction execution. Utilizing the atomic `action` construct in the IVy language, action `update_Ri` requires that all actions within itself fully execute before another reaction can occur. A comprehensive description of IVy's language syntax and semantics, as well as the IVy verification tool can be found in [31,27,26].

7 Shortest Trace Generation

This section introduces the dependency graph for a CRN and describes how it either proves unreachability of a rare event or guides the shortest desirable trace generation by automated assertion creation and insertion in the IVy model, which is used to rapidly enumerate many such traces through compositional testing.

Dependency Graph for Shortest Trace Generation. Denote the edge $\rightsquigarrow$ as the *dependency relation* between two reactions, which is a binary relation defined in Definition 3. $\mathcal{R}_i \rightsquigarrow \mathcal{R}_j$ indicates that $\mathcal{R}_i$ *depends on* $\mathcal{R}_j$. Denote $\rightsquigarrow^+$ as the transitive closure of the dependency relation and $\mathcal{R}_i \rightsquigarrow^+ \mathcal{R}_j$ means that there is a path of dependency edges from $\mathcal{R}_i$ to $\mathcal{R}_j$: $\mathcal{R}_i \rightsquigarrow \cdots \rightsquigarrow \mathcal{R}_k \rightsquigarrow \cdots \rightsquigarrow \mathcal{R}_j$.

Dependency relationships are established based on the minimum number of times a reaction must execute for the model to reach a target state. For consistency, create an abstract reaction $\mathcal{R}_\Psi$ representing the target specification Ψ such that $\Delta(\mathcal{R}_\Psi) = C_\Psi - s_0(\mathcal{X}_\Psi)$ as described in Definition 1. Informally, $\Delta(\mathcal{R}_\Psi)$ represents the difference between the target and initial values for the abstract species $\mathcal{X}_\Psi$, and $\mathcal{R}_\Psi$ is required to execute at least $\Delta(\mathcal{R}_\Psi)$ times to achieve the target value. This provides a starting point for dependency graph construction.

Definition 1 (Minimum Required Reaction Executions). *The minimum required reaction executions $\Delta : \mathcal{R} \to \mathbb{Z}$ maps a reaction $\mathcal{R}_i$ in the set of reactions $\mathcal{R}$ of a CRN $\mathcal{M}$ to the minimum number of times $\mathcal{R}_i$ must execute for $\mathcal{M}$ to reach a target state s_Ψ from its initial state s_0.*

Definition 2 (Enabled and Disabled Reaction Sets). *The reaction set $\mathcal{R}$ is partitioned into two subsets: set E containing only reactions $\mathcal{R}_i$ enabled to execute $\Delta(\mathcal{R}_i)$ times from s_0, and set D containing all other reactions. As such, $E \cup D = \mathcal{R}$ and $E \cap D = \emptyset$.*

Definition 3 (Dependency Relation). *A dependency relation with respect to a CRN $\mathcal{M} = \langle \mathcal{X}, \mathcal{R}, s_0 \rangle$ is an antireflexive binary relation $\rightsquigarrow \subseteq D \times \mathcal{R}$. $\mathcal{R}_i$ depends on $\mathcal{R}_j$, denoted as $\mathcal{R}_i \rightsquigarrow \mathcal{R}_j$, iff all conditions below hold:*
1. $\mathcal{R}_i \in D \wedge (\mathcal{R}_i \neq \mathcal{R}_j) \wedge \neg(\mathcal{R}_j \rightsquigarrow^+ \mathcal{R}_i)$,
2. $(\Delta(\mathcal{R}_i) > 0 \wedge \mathsf{Reactant}_i \cap \mathsf{Product}_j \neq \emptyset) \vee$
   $(\Delta(\mathcal{R}_i) < 0 \wedge \mathsf{Product}_i \cap \mathsf{Reactant}_j \neq \emptyset)$, *and*
3. $(\exists \mathcal{R}_k \text{ s.t. } \mathcal{R}_j \rightsquigarrow \mathcal{R}_k) \vee (\mathcal{R}_j \in E)$.

The conditions of Definition 3 are described intuitively as follows:
1. $\mathcal{R}_i$ must be initially disabled in order to depend on another reaction and it cannot depend on itself directly or cyclically.
2. $\mathcal{R}_i$ must execute a nonzero number of times. If $\mathcal{R}_i$ requires the production (consumption) of a species, $\mathcal{R}_j$ must produce (consume) that species.
3. $\mathcal{R}_j$ must depend on another reaction, or the initial state must supply abundant species counts to enable $\mathcal{R}_j$ to execute $\Delta(\mathcal{R}_j)$ times.

An algorithm for dependency graph construction first explores Conditions 1 and 2 before removing dependency relations that fail Condition 3. Note that $\Delta(\mathcal{R}_i)$ can be negative in $\mathcal{R}_\Psi$, indicating that if a target species count is less than its initial count, it is desirable to find reactions that consume, rather than produce, the required species. Cyclic dependencies are not permitted under Definition 3. This addresses the paradox in which a reaction must execute an infinite number of times to produce enough of a species to enable itself to execute. These conditions are crucial in proving unreachability, as described later in this section.

The dependency relation between all reaction pairs guides the construction of a dependency graph, which is a directed graph with vertices representing reactions and edges representing dependency relation. The root of the graph is $\mathcal{R}_\Psi$, as no other reaction can depend on it. Algorithm 1 outlines the procedure for recursively constructing the graph following Definitions 1, 2, and 3. Figure 3a shows a case for the motivating example. Starting with $\mathcal{R}_\Psi$, establish the relation

Algorithm 1 Dependency Graph Construction

Require: $\mathcal{M} = \langle \mathcal{X}, \mathcal{R}, s_0 \rangle,\ \Psi$.

procedure MAIN
  Generate $\mathcal{R}_\Psi$ and calculate $\Delta(\mathcal{R}_\Psi) = C_\Psi - s_0(\mathcal{X}_\Psi)$
  Initialize all dependency relations to `false`
  BUILDGRAPH($\mathcal{R}_\Psi$)
  if there exists $\mathcal{R}_i$ such that $\mathcal{R}_\Psi \rightsquigarrow \mathcal{R}_i$ **then** begin trace generation.
  else Ψ is unreachable; terminate.

procedure BUILDGRAPH(Reaction $\mathcal{R}_i$)
  if $\mathcal{R}_i \in E$ **then return**
  for all $\mathcal{R}_j$ such that $\neg(\mathcal{R}_i = \mathcal{R}_j \vee \mathcal{R}_j \rightsquigarrow^+ \mathcal{R}_i)$ **do**
    $\delta := \Delta(\mathcal{R}_i) - \min_{\mathcal{X}_k \in \mathsf{Reactant}_i}(s_0(\mathcal{X}_k))$
    if $\delta > 0$ **then** $\Delta(\mathcal{R}_j) := \Delta(\mathcal{R}_j) + \delta$; $\mathcal{R}_i \rightsquigarrow \mathcal{R}_j :=$ `true`
    BUILDGRAPH($\mathcal{R}_j$)
    if $\neg((\exists \mathcal{R}_k \text{ s.t. } \mathcal{R}_j \rightsquigarrow \mathcal{R}_k) \vee (\mathcal{R}_j \in E))$ **then** $\mathcal{R}_i \rightsquigarrow \mathcal{R}_j :=$ `false`

$\mathcal{R}_\Psi \rightsquigarrow \mathcal{R}_5$ since only $\mathcal{R}_5$ can produce G_{bg} required by $\mathcal{R}_\Psi$. The relations $\mathcal{R}_5 \rightsquigarrow \mathcal{R}_3$ and $\mathcal{R}_5 \rightsquigarrow \mathcal{R}_8$ are similarly established. Since both $\mathcal{R}_3$ and $\mathcal{R}_8$ are enabled in s_0, the algorithm terminates without removing any relations. Figure 3b, in contrast, shows an unreachable target state assuming a different s_0, where it fails to supply sufficient molecules of G, e.g., $s_0(G) = 1$. Thus, relations $\mathcal{R}_5 \rightsquigarrow \mathcal{R}_3$, $\mathcal{R}_5 \rightsquigarrow \mathcal{R}_8$, and $\mathcal{R}_5 \rightsquigarrow \mathcal{R}_7$ are established. Since $\mathcal{R}_7 \notin E$, $\mathcal{R}_7 \rightsquigarrow \mathcal{R}_6$ and $\mathcal{R}_7 \rightsquigarrow \mathcal{R}_5$ are established, creating cyclic dependency $\mathcal{R}_5 \rightsquigarrow^+ \mathcal{R}_5$. Therefore, $\mathcal{R}_7 \rightsquigarrow \mathcal{R}_5$ is disestablished. Because it is impossible to produce enough G_{bg} to execute $\mathcal{R}_7$, $\mathcal{R}_5 \rightsquigarrow \mathcal{R}_7$ is disestablished. Similarly, $\mathcal{R}_\Psi \rightsquigarrow \mathcal{R}_5$ must be disestablished, leaving only $\mathcal{R}_\Psi$ in the dependency graph, proving unreachability of Ψ.

The derivation of a dependency graph enables three crucial developments. First, it enables CRN reachability analysis as described in Theorem 1. Second, a set of desirable reactions enables the construction of traces leading to a target state. This is useful when very few traces reach a target state. Lastly, shortest traces (by reaction execution count) are obtainable as shown in Theorem 2.

Theorem 1 (Unreachable Target). *In a dependency graph constructed per Definitions 1, 2, and 3 for a CRN $\mathcal{M} = \langle \mathcal{X}, \mathcal{R}, s_0 \rangle$, if $\mathcal{R}_\Psi \in D \wedge (\forall \mathcal{R}_i \in \mathcal{R}.\ \neg(\mathcal{R}_\Psi \rightsquigarrow^+ \mathcal{R}_i \wedge \mathcal{R}_i \in E)$, any target state s_Ψ is unreachable from s_0.*

Proof (Theorem 1). Assume the following opposite statement holds: A dependency graph constructed with Definitions 1, 2, and 3 for a CRN satisfies $\mathcal{R}_\Psi \in D \wedge (\forall \mathcal{R}_i \in \mathcal{R}.\ \neg(\mathcal{R}_\Psi \rightsquigarrow^+ \mathcal{R}_i \wedge \mathcal{R}_i \in E)$ and some target state s_Ψ is reachable. In a closed CRN, if s_Ψ is reachable, then either (1) $\mathcal{R}_\Psi \in E$ in s_0 or (2) $\mathcal{R}_\Psi \in D$ in s_0 and there exists a reaction sequence, say $\mathcal{R}_i \ldots \mathcal{R}_l$ and $\mathcal{R}_i \in E$, that enables $\mathcal{R}_\Psi$ in a future state. Case (1) is trivially true for this theorem. For case (2), construct a dependency graph by Definition 3. Then it must hold that $\exists \mathcal{R}_i \in \mathcal{R}.\ \mathcal{R}_\Psi \rightsquigarrow^+ \mathcal{R}_i \wedge \mathcal{R}_i \in E$. Because $\mathcal{R}_i \ldots \mathcal{R}_l$ exists to enable $\mathcal{R}_\Psi$, the constructed dependency graph must establish $\mathcal{R}_\Psi \rightsquigarrow^+ \mathcal{R}_i$ (Condition 2) and the

terminal reaction $\mathcal{R}_i$ for $\mathcal{R}_\Psi \rightsquigarrow^+ \mathcal{R}_i$ must be enabled in s_0 (Condition 3) when $\mathcal{R}_\Psi \rightsquigarrow^+ \mathcal{R}_i$ is acyclic (Condition 1). This contradicts the assumption. □

Grouping dependency relations into weighted branches as described in Definition 4 enables the discovery of trace lengths. Intuitively, a weighted branch $\mathcal{B}$ is a weighted path $\mathcal{R}_\Psi \rightsquigarrow^+ \mathcal{R}_i \wedge \mathcal{R}_i \in E$ in the dependency graph. The branch weight $W(\mathcal{B})$ is the minimal length of a trace including only the reactions in $\mathcal{B}$.

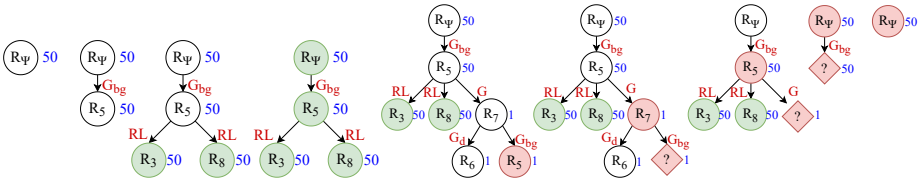


Fig. 3: Dependency Graph Examples.

Definition 4 (Weighted Branch on a Dependency Graph). *Let a branch* $\mathcal{B}_\alpha = \{\mathcal{R}_i \mid (\mathcal{R}_\Psi \rightsquigarrow^+ \mathcal{R}_i \rightsquigarrow^+ \mathcal{R}_j \wedge \mathcal{R}_j \in E) \vee (\mathcal{R}_\Psi \rightsquigarrow^+ \mathcal{R}_i \wedge \mathcal{R}_i \in E)\}$. *The weight of* $\mathcal{B}_\alpha$ *is* $W(\mathcal{B}_\alpha) = \sum_{\mathcal{R}_i \in \mathcal{B}_\alpha} \Delta(\mathcal{R}_i)$, *which is the sum of the minimal number of reaction executions when executing only the reactions in* $\mathcal{B}_\alpha$. *Label a branch with* $L(\mathcal{B}_\alpha) = \{\mathsf{Reactant}_i \cap \mathsf{Product}_j \mid \mathcal{R}_i \rightsquigarrow \mathcal{R}_j \wedge \mathcal{R}_i, \mathcal{R}_j \in \mathcal{B}_\alpha\}$.

Theorem 2 (Shortest Trace). *Construct a set of branches* $\mathbb{B}_x$ *such that each branch is required to produce a unique species and* $\forall\, \mathbb{B}_y \neq \mathbb{B}_x$, $\sum_{\mathcal{B}_a \in \mathbb{B}_x} W(\mathcal{B}_a) \leq \sum_{\mathcal{B}_b \in \mathbb{B}_y} W(\mathcal{B}_b)$. *Let* $\rho(\mathbb{B}_x) = \bigcup_{\mathcal{B}_a \in \mathbb{B}_x} \mathcal{R}_i \in \mathcal{B}_a$. *There exists some sequence of reactions in* $\rho(\mathbb{B}_x)$, *where each* $\mathcal{R}_i \in \rho(\mathbb{B}_x)$ *is executed precisely* $\Delta(\mathcal{R}_i)$ *times such that the resulting trace yields the minimal reaction execution count.*

Intuitively, a branch is a set of reactions that together produce or consume at least one desired species for $\mathcal{R}_\Psi$. In the motivating example shown in Figure 3a, there are two branches: $\{\mathcal{R}_5, \mathcal{R}_3\}$ and $\{\mathcal{R}_5, \mathcal{R}_8\}$. Shortest traces are generated by first gathering the non-empty lowest-weighted set of branches producing unique required species. For example, $\mathcal{R}_3$ and $\mathcal{R}_8$ produce the same species needed by $\mathcal{R}_\Psi$, so only one branch is required to be included in $\mathbb{B}_x$, but both $\mathcal{R}_3$ and $\mathcal{R}_8$ can be included in $\mathbb{B}_x$ since $\Delta(\mathcal{R}_3) = \Delta(\mathcal{R}_8)$. If another branch produced a different required species, it would be included in $\mathbb{B}_x$ with $\mathcal{R}_3$ and/or $\mathcal{R}_8$.

Proof (Theorem 2). Consider a trace σ_η containing n_η reaction executions produced by the method in Theorem 2. Consider a trace σ_β containing n_β reaction executions, such that $n_\beta < n_\eta$. Because σ_β reaches a target state from the initial state, σ_β must include reactions that are required to reach a target state from the initial state. These actions are, by definition, included in a branch in the dependency graph. Because each reaction $\mathcal{R}_i$ in σ_β is, by definition, required to

execute at least $\Delta(\mathcal{R}_i)$ times, the branches used to derive sequence σ_β from the dependency graph must have a lesser cumulative weight than the branches used to derive sequence σ_η. However, in constructing sequence σ_η, only the branches with the least cumulative weight are selected. Thus, either σ_β is of the same length as σ_η (causing both traces to be shortest traces), or σ_β cannot exist, proving by contradiction that a trace σ_η derived by the procedure in Theorem 2 must be a trace with the fewest total reaction executions. □

Shortest Trace Generation using Compositional Testing. After obtaining $\mathbb{B}_x$ as prescribed by Theorem 2, we exclude reactions $\mathcal{R}_l \notin \mathbb{B}_x$ as they do not directly or indirectly contribute to reaching s_Ψ. Each remaining reaction $\mathcal{R}_i \in \mathbb{B}_x$ is then assigned a unique variable `Ri_executions` in `protocol` corresponding to the number of times $\mathcal{R}_i$ has been executed. A shortest trace can then be obtained via randomized compositional testing by asserting `Ri_executions` $< \Delta(\mathcal{R}_i)$ as a precondition for `update_Ri`. In Figure 3a, $\rho(\mathbb{B}_x) = \{\mathcal{R}_3, \mathcal{R}_5, \mathcal{R}_8\}$, and therefore, all other reactions are removed. Assertions are then added to ensure each reaction only executes the minimum required number of times needed to reach s_Ψ. For example, assertion `R5_executions` < 50 is added to `update_R5` to ensure that $\mathcal{R}_5$ does not execute more than 50 times. During compositional testing of the triple $\langle\alpha\rangle p_1 \langle\alpha\rangle$, p_1 is the `protocol` object in a CRN model and α is `Ri_executions` $< \Delta(\mathcal{R}_i)$. This assertion serves as a part of the precondition for action `update_Ri` defined in p_1. It becomes an assumption when testing p_1 locally. Execution of every $\mathcal{R}_i \in \rho(\mathbb{B}_x)$ starts with action `update_Ri`, which is called by p_1's environment process p_2. $\mathcal{R}_i$ cannot occur if `Ri_executions` has already reached $\Delta(\mathcal{R}_i)$, indicating that the minimal number of the required $\mathcal{R}_i$'s action sequences to reach s_Ψ has already occurred, even if $\mathcal{R}_i$ is enabled. The same assertion is inserted as a part of the precondition for the `check_guard_Ri` action defined in the `inspector` object, which is part of p_2. It becomes a guarantee during the local testing of p_1 because `check_guard_Ri` is p_1's output action. Inserting the assertion `Ri_executions` $< \Delta(\mathcal{R}_i)$ in this way guarantees that only the minimum number of every $\mathcal{R}_i \in \rho(\mathbb{B}_x)$ can appear in the resulting trace σ, whose length is therefore, the shortest. Note that only once the contributing reactions (i.e., those in $\mathbb{B}_x$) have been established from the dependency graph is the IVy model actually constructed with all of the aforementioned assertions.

8 Generation of Diverse Traces

To optimize the chance of finding representative traces leading to rare-event states with relatively high probability, it is desirable to obtain a diverse assortment of traces. Using reactions from equally weighted branches and sequence prefixes can generate a diverse set of traces from compositional testing.

Equally Weighted Sets of Branches. Given $\mathbb{B}_x = \{\mathcal{B}_a, \mathcal{B}_b\}$, $\mathcal{R}_\Psi \rightsquigarrow^+ \mathcal{R}_i \rightsquigarrow^+ \mathcal{R}_x$, and $\mathcal{R}_\Psi \rightsquigarrow^+ \mathcal{R}_i \rightsquigarrow^+ \mathcal{R}_y$, if a species $\mathcal{X}_i \in \mathsf{Reactant}_i$ is produced equally

effectively by a reaction $\mathcal{R}_y \in \mathcal{B}_b$ as by a reaction $\mathcal{R}_x \in \mathcal{B}_a$, in order to produce more diverse traces, the traces should include both $\mathcal{R}_x$ and $\mathcal{R}_y$, but they should be executed at a lower frequency than that of other reactions, in order to not favor the production of one species more than another during compositional testing. To accomplish this, let each reaction be assigned a *tier value*, where $tier(\mathcal{R}_\alpha) = |\{\mathcal{R}_i \in \mathcal{B} \mid (\mathcal{R}_i \rightsquigarrow^+ \mathcal{R}_\alpha\}|$ and let the *frequency* of $\mathcal{R}_\beta$ be denoted as $freq(\mathcal{R}_\beta) = |\{\mathcal{R}_\alpha \mid tier(\mathcal{R}_\alpha) = tier(\mathcal{R}_\beta) \wedge \mathcal{R}_\alpha, \mathcal{R}_\beta \in \rho(\mathbb{B}_x)\}|^{-1}$. The CRN model is then modified as follows: (1) The assertion (`Rx_executions` + `Ry_executions`) $< \Delta(\mathcal{R}_y)$ is inserted as a precondition for both $\mathcal{R}_x$ and $\mathcal{R}_y$; and (2) `selector` is modified to let $\mathcal{R}_x, \mathcal{R}_y$ execute with a frequency of $freq(\mathcal{R}_y)$. These modifications ensure that the generated traces are diverse and the shortest. For instance, for $\mathcal{R}_3$ and $\mathcal{R}_8$ in our motivating example, because $L(\mathcal{R}_5 \rightsquigarrow \mathcal{R}_3) = L(\mathcal{R}_5 \rightsquigarrow \mathcal{R}_8) = \mathrm{RL}$, $\mathcal{R}_3$ and $\mathcal{R}_8$ have the same tier value, and $\Delta(\mathcal{R}_3) = \Delta(\mathcal{R}_8) = 50$, we change the precondition assertions for `update_R3` and `update_R8` to be (`R3_executions` + `R8_executions`) < 50. We also modify `selector` so that $\mathcal{R}_3$ and $\mathcal{R}_8$ execute once per every two times they are enabled.

Sequence Prefixes for Generating Diverse Traces. Due to the highly concurrent nature of CRNs, multiple unique weighted branch sets can create shortest traces according to Theorem 2. In the motivating example, the branches $\{\mathcal{R}_5, \mathcal{R}_3\}$ and $\{\mathcal{R}_5, \mathcal{R}_8\}$ each have a weight of 100. Executing a total of 100 reactions using a combination of $\mathcal{R}_5$, $\mathcal{R}_3$, and $\mathcal{R}_8$ as prescribed by the dependency graph results in the generation of a shortest trace. Dependency information is used to partition the state space into prefix-based subsets. For example, all traces generated using branches $\{\mathcal{R}_5, \mathcal{R}_3\}$ and $\{\mathcal{R}_5, \mathcal{R}_8\}$ must begin with the sequence prefix $\mathcal{R}_8, \mathcal{R}_5$; $\mathcal{R}_3, \mathcal{R}_5$; $\mathcal{R}_3, \mathcal{R}_8$; or $\mathcal{R}_8, \mathcal{R}_3$. Because each prefix is unique, there is no need to check for duplicate traces except among traces with identical prefixes. Therefore, it is possible to discard saved trace information after each prefix set is completed, saving only the sum of the probability of each trace. This saves time and allows for more trace enumeration using less memory. This state space partitioning method is achieved by using stochastic simulation to obtain the state immediately following each prefix. This state is then used in lieu of the model's initial state for compositional testing, and the number of required executions for each reaction in the prefix decrements. The prefix reaction sequence is prepended to each trace before stochastic simulation to obtain the trace's probability.

9 Tool Implementation

Ragtimer tool flow. We have implemented the proposed techniques in a prototype tool called Ragtimer, so named because it is designed to be fast and carefree for a user, much like the Ragtime musical genre. As Figure 1 shows, Ragtimer first builds a dependency graph from the CRN model and its rare-event reachability property. The dependency graph is then used to determine reachability of the target rare-event specification and it either provides a proof of unreachability, or creates an IVy model containing only reactions in the set of

branches with the lowest weight, as described in Theorem 2 and Section 8. To optimize the chance of a diverse set of representative traces to reach the target rare-event, RAGTIMER partitions the possible traces by shortest trace prefixes. Using information obtained from the dependency graph analysis, it automatically creates and then inserts assertions into the IVy model. RAGTIMER then invokes randomized compositional testing to produce a user-specified quantity of unique traces to the target state. Lastly, it simulates each trace to obtain its rare-event reachability probability by interfacing the PRISM probabilistic model checker [22] and sums them together to obtain the total probability of the set of traces.

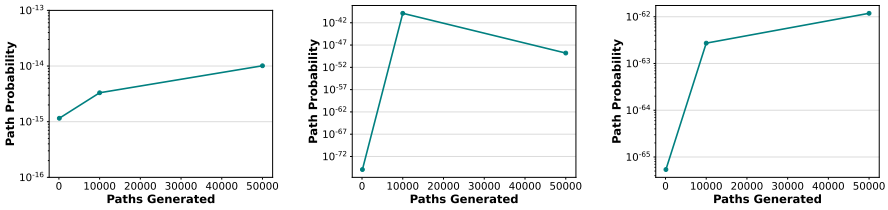
Probability acquisition. The main advantage of RAGTIMER is its effectiveness in rapidly generating a large number of the shortest desirable traces to a rare-event of interest. The lower bound for the rare-event probability $\mathsf{P}_{min}(\Diamond^{[0,T]}\ \Psi)$, where Ψ is the rare-event in the form of $\mathcal{X}_i = C$, is obtained by summing up the probabilities of all generated traces. RAGTIMER interfaces the stochastic simulation engine in PRISM to obtain probabilities for all individual traces. When branch prefixes are used for trace generation, this simulation occurs in batches, where each batch is for one equally weighted prefix from the dependency graph. An alternative approach is to construct a partial state space from the generated traces first and then interface a probabilistic model checking tool to obtain $\mathsf{P}_{min}(\Diamond^{[0,T]}\ \Psi)$. However, such construction would require an equivalent simulation effort for each trace in addition to state duplication. Our experiments showed that state space construction yielded an identical lower probability bound to trace simulation, but it used significantly more computational resources.

10 Results and Discussion

We obtained all results on a machine with an AMD Ryzen Threadripper 12-Core 3.5 GHz Processor and 132 GB of RAM, running Ubuntu Linux (v18.04.3). With one CPU and 16 GB of RAM allocated, RAGTIMER was tested on three representative CRN models. The rarity of their target specifications combined with the large state spaces can quickly overwhelm computational resources. Reaction propensities for all models presented in this section are in molecules per second.

Single species production-degradation model. This model consists of two species reacting through a production-degradation interaction [19]: $\mathcal{R}_1 : S_1 \xrightarrow{1.0} S_1 + S_2, \mathcal{R}_2 : S_2 \xrightarrow{0.025} \emptyset$. The initial molecule count for species vector (S_1, S_2) forms the initial state: $s_0 = [1, 40]$. The desired CSL property of this model is $\mathsf{P}_{=?}(\Diamond^{[0,100]}\, S_2 = 80)$. Obviously, the the shortest trace to s_Ψ is simply a repetition of $\mathcal{R}_1$. RAGTIMER quickly discovers this shortest trace, and alerts the user that only one shortest trace can be generated. Restriction is then loosened in the IVy model to allow extraneous reactions, i.e., $\mathcal{R}_2$, to randomly execute at a desired frequency to produce additional traces. Figure 4a shows the probability increase when these traces are generated. Generating $10,000$ additional traces

repeatedly took less than 2 minutes. This shows the value of controlling reaction restrictions. While obtaining a single shortest trace can be valuable, allowing many traces with extraneous reactions enables the accumulation of increased probability. In our experiments, when these probabilities begin to converge, it becomes helpful to increase the allowance for extraneous reactions to generate more diverse traces.



(a) Single species (b) Enzymatic futile cycle (c) Modified yeast polariz.

Fig. 4: Cumulative probability of desirable traces produced by RAGTIMER.

Enzymatic futile cycle model. This example models the enzymatic futile cycle motif consisting of two single-substrate enzymatic reaction scheme, one transforming S_2 into S_5 catalyzed by S_1 and the other transforming S_5 into S_2 catalyzed by S_4 [19]:

$$\mathcal{R}_1 : \; S_1 + S_2 \xrightarrow{1.0} S_3, \quad \mathcal{R}_2 : \; S_3 \xrightarrow{1.0} S_1 + S_2, \quad \mathcal{R}_3 : \; S_3 \xrightarrow{0.1} S_1 + S_5,$$
$$\mathcal{R}_4 : \; S_4 + S_5 \xrightarrow{1.0} S_6, \quad \mathcal{R}_5 : \; S_6 \xrightarrow{1.0} S_4 + S_5, \quad \mathcal{R}_6 : \; S_6 \xrightarrow{0.1} S_4 + S_2.$$

This motif widely exists in biological systems including GTPase cycles, MAPK cascades, and glucose mobilization [37]. The initial molecule count for species vector $(S_1, S_2, S_3, S_4, S_5, S_6)$ forms the initial state: $s_0 = [1, 50, 0, 1, 50, 0]$. The rare-event property is $\mathsf{P}_{=?}(\Diamond^{[0,100]}\, S_5 = 25)$. Similar to the single-species model, this model only produces one shortest trace that alternates between $\mathcal{R}_4$ and $\mathcal{R}_6$. Figure 4b shows the probability trend with additional traces. Since these traces each reach a rare-event state, it enables a probability increase of over 30 orders of magnitude using under 2 minutes of total run time. Notably, the test with $50,000$ traces yielded a much lower probability than the test with $10,000$ traces, which demonstrates the importance of the user-specified frequency to execute extraneous reactions. In the test with $50,000$ traces, extraneous reactions were allowed to execute more frequently, resulting in slightly longer traces with a shorter runtime per trace production. However, they lowered the overall probability despite the presence of more traces.

Modified yeast polarization model. RAGTIMER is well-suited for mining rare-event traces for large-state models such as the modified yeast polarization

model introduced in Section 4. When simulating this model using the standard *stochastic simulation algorithm* (SSA) implemented in the PRISM probabilistic model checking tool, the average trace length is generally over 4,000 reactions before reaching a target state, and the total probability for over 500,000 traces is rounded to 0 due to floating-point precision limitations. That is, SSA alone generates paths with probabilities much lower than 4.9×10^{-324}. On the contrary, RAGTIMER can find a collection of short and generally more probable traces. Figure 4c shows that with only 10,000 traces that RAGTIMER found within 2 minutes, their cumulative probability was already significant. Given its run time is not significantly different from pure SSA per generated trace, RAGTIMER is a more effective and efficient first strategy for rare-event analysis than SSA.

When this model is instead initialized with $G = 49$, the target state becomes unreachable. RAGTIMER quickly detected this unreachability and terminated, while other tools, including `modes` in the MODEST TOOLSET and SSA in PRISM, executed indefinitely attempting to simulate traces. When tested on unreachable variants of the modified yeast polarization and futile cycle models, RAGTIMER reported unreachability in under 1 second and did not attempt to generate traces.

Table 1: Rare-event simulation results for the three examples using `modes`.

	single species	enzymatic futile cycle	modified yeast polar.
rare-event probability	3.0631×10^{-7}	1.7043×10^{-7}	1.7002×10^{-6}
runtime (seconds)	5.7	127.9	26.9
peak memory (MB)	143	141	144

Comparison to `modes` rare-event simulation engine. The `modes` tool was able to efficiently and accurately compute rare-event probabilities. Accuracy means the closeness of reported rare-event probabilities between the results from `modes`, shown in Table 1, and [19] for the single-species prodcuction-degradation and enzymatic futile cycle models, and [9] for the modified yeast polarization model. However, the compositional importance function required for rare-event simulation poses a major limitation on global variables shared among multiple components, each representing a reaction in the CRN. While manual modifications to the model's importance function can be made to get around this issue, it requires user intervention and a thorough understanding of the CRN model and the MODEST language, with the possibility of introducing errors. `modes` was also tested on the modified yeast polarization model initialized with $G = 49$ to produce an unreachable target. Despite its speed for verification of reachable models, it ran for 24 hours on 23 CPU threads, performing over 1 million sweeps of the model without determining the target specification is unreachable.

Comparison to probabilistic model checking tools. After bounding all species' molecule counts to a reasonably large range of $[0, 150]$, we attempted to verify the modified yeast polarization model using the STORM probabilistic model checker [13] with the SYLVAN library [8] to allow for parallel construction of the symbolic state space. Although symbolic state space construction completed quickly, STORM failed to complete the transient analysis after running for 30 days. This is due to the overhead in converting symbolic state space to the sparse matrix representation to perform time-bounded transient analysis. Similar to STORM, PRISM was also unable to complete probabilistic model checking of this property within a reasonable time bound. However, running the state-truncation probabilistic model checker STAMINA [32] on the same model produced a probability bound of $[1.64 \times 10^{-6}, 23.01 \times 10^{-6}]$ after two days.

11 Conclusion

This paper presents a scalable and fully automated approach to rapidly enumerate a large number of traces guaranteed to reach desirable rare-event states for a given CRN model. It includes both a layered and service-oriented compositional modeling method and a dependency graph analysis technique to either prove unreachability or guide the generation of a variety of the shortest traces. Together, they automatically construct assumptions and guarantees that enable compositional testing to produce many desirable traces, which are then simulated to provide the lower probability bound for the rare-event. Efficiency in both trace generation and rare-event probability computation is demonstrated in three challenging CRN models. For future work, we will investigate effective cycle detection method to further improve rare-event trace generation.

Acknowledgment We thank Arnd Hartmanns (U. Twente) for his help with `modes`; Chris Winstead (Utah State U.), Chris Myers (U. Colorado Boulder), and Hao Zheng (U. South Florida) for their feedback. This work was supported by the National Science Foundation under Grant No. 1856733. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

1. SBML-to-PRISM translator, http://www.prismmodelchecker.org/sbml/
2. Adelman, J.L., Grabe, M.: Simulating rare events using a weighted ensemble-based string method. The Journal of Chemical Physics **138**(4), 044105 (2013). https://doi.org/10.1063/1.4773892, https://doi.org/10.1063/1.4773892
3. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time markov chains. ACM Trans. Comput. Logic **1**(1), 162–170 (Jul 2000)
4. Budde, C.E., D'Argenio, P.R., Hartmanns, A.: Automated compositional importance splitting. Science of Computer Programming **174**, 90–108 (2019). https://doi.org/https://doi.org/10.1016/j.scico.2019.01.006, https://www.sciencedirect.com/science/article/pii/S0167642318301503
5. Budde, C.E., Hartmanns, A.: Replicating RESTART with prolonged retrials: An experimental report. In: Groote, J.F., Larsen, K.G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12652, pp. 373–380. Springer (2021). https://doi.org/10.1007/978-3-030-72013-1_21, https://doi.org/10.1007/978-3-030-72013-1_21
6. Chellaboina, V., Bhat, S.P., Haddad, W.M., Bernstein, D.S.: Modeling and analysis of mass-action kinetics. IEEE Control Systems Magazine **29**(4), 60–78 (2009)
7. Daigle, B.J.J., Roh, M.K., Gillespie, D.T., Petzold, L.R.: Automated estimation of rare event probabilities in biochemical systems. J Chem Phys **134**(4), 044110 (Jan 2011). https://doi.org/10.1063/1.3522769
8. Dijk, T., Pol, J.: Sylvan: Multi-core framework for decision diagrams. Int. J. Softw. Tools Technol. Transf. **19**(6), 675–696 (nov 2017). https://doi.org/10.1007/s10009-016-0433-2, https://doi.org/10.1007/s10009-016-0433-2
9. Donovan, R.M., Sedgewick, A.J., Faeder, J.R., Zuckerman, D.M.: Efficient stochastic simulation of chemical kinetics networks using a weighted ensemble of trajectories. The Journal of Chemical Physics **139**(11), 115105 (Sep 2013). https://doi.org/10.1063/1.4821167
10. Drawert, B., Lawson, M.J., Petzold, L., Khammash, M.: The diffusive finite state projection algorithm for efficient simulation of the stochastic reaction-diffusion master equation. The Journal of Chemical Physics **132**(7), 074101 (2010). https://doi.org/10.1063/1.3310809, https://doi.org/10.1063/1.3310809
11. Giannakopoulou, D., Pasareanu, C., Blundell, C.: Assume-guarantee testing for software components. Software, IET **2**, 547 – 562 (01 2009). https://doi.org/10.1049/iet-sen:20080012
12. Hartmanns, A., Hermanns, H.: The Modest Toolset: An integrated environment for quantitative modelling and verification. In: Ábrahám, E., Havelund, K. (eds.) TACAS. LNCS, vol. 8413, pp. 593–598. Springer (2014)
13. Hensel, C., Junges, S., Katoen, J.P., Quatmann, T., Volk, M.: The probabilistic model checker Storm. International Journal on Software Tools for Technology Transfer **24**(4), 589–610 (Aug 2022). https://doi.org/10.1007/s10009-021-00633-z
14. Jegourel, C., Legay, A., Sedwards, S.: Cross-entropy optimisation of importance sampling parameters for statistical model checking. In: Proceedings of the 24th international conference on Computer Aided Verification. pp. 327–342. CAV'12, Springer-Verlag, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31424-7_26, http://dx.doi.org/10.1007/978-3-642-31424-7_26

15. Jegourel, C., Legay, A., Sedwards, S.: Importance splitting for statistical model checking rare properties. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification. pp. 576–591. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
16. Kahn, H.: Random sampling (monte carlo) techniques in neutron attenuation problems–I. Nucleonics **6**(5), 27; passim (May 1950)
17. Kahn, H., Marshall, A.W.: Methods of reducing sample size in monte carlo computations. Journal of the Operations Research Society of America **1**(5), 263–278 (1953). https://doi.org/10.1287/opre.1.5.263, https://doi.org/10.1287/opre.1.5.263
18. Kahn, H., Harris, T.E.: Estimation of particle transmission by random sampling. National Bureau of Standards applied mathematics series **12**, 27–30 (1951)
19. Kuwahara, H., Mura, I.: An efficient and exact stochastic simulation method to analyze rare events in biochemical systems. The Journal of Chemical Physics **129**(16), 165101 (Oct 2008). https://doi.org/10.1063/1.2987701
20. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic Model Checking, pp. 220–270. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
21. Kwiatkowska, M., Norman, G., Parker, D.: Using probabilistic model checking in systems biology. SIGMETRICS Perform. Eval. Rev. **35**(4), 14–21 (mar 2008). https://doi.org/10.1145/1364644.1364651, https://doi.org/10.1145/1364644.1364651
22. Kwiatkowska, M., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: Proceedings of the 23rd International Conference on Computer Aided Verification. pp. 585–591. CAV'11, Springer-Verlag, Berlin, Heidelberg (2011)
23. L'Ecuyer, P., LeGland, F., Lezaud, P., Tuffin, B.: Splitting techniques (Jan 2009)
24. Legay, A., Lukina, A., Traonouez, L.M., Yang, J., Smolka, S.A., Grosu, R.: Statistical Model Checking, pp. 478–504. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-319-91908-9_23, https://doi.org/10.1007/978-3-319-91908-9_23
25. McMillan, K.: Modular specification and verification of a cache-coherent interface. In: Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design. pp. 109–116. FMCAD '16, FMCAD Inc, Austin, Texas (2016)
26. McMillan, K.L.: IVy: http://microsoft.github.io/ivy/ (2019), https://github.com/kenmcmil/ivy
27. McMillan, K.L., Padon, O.: Ivy: A multi-modal verification tool for distributed algorithms. In: Lahiri, S.K., Wang, C. (eds.) Computer Aided Verification. pp. 190–202. Springer International Publishing, Cham (2020)
28. McMillan, K.L., Zuck, L.D.: Compositional testing of internet protocols. In: 2019 IEEE Cybersecurity Development (SecDev). pp. 161–174 (Sep 2019). https://doi.org/10.1109/SecDev.2019.00031
29. Myers, C.J.: Engineering Genetic Circuits. Chapman & Hall/CRC Mathematical and Computational Biology, Chapman & Hall/CRC, 1 edn. (July 2009)
30. Okamoto, M.: Some inequalities relating to the partial sum of binomial probabilities. Annals of the Institute of Statistical Mathematics **10**(1), 29–35 (1959). https://doi.org/10.1007/BF02883985, https://doi.org/10.1007/BF02883985
31. Padon, O., McMillan, K.L., Panda, A., Sagiv, M., Shoham, S.: Ivy: Safety verification by interactive generalization. SIGPLAN Not. **51**(6), 614–630 (jun 2016). https://doi.org/10.1145/2980983.2908118, https://doi.org/10.1145/2980983.2908118

32. Roberts, R., Neupane, T., Buecherl, L., Myers, C.J., Zhang, Z.: STAMINA 2.0: Improving scalability of infinite-state stochastic model checking. In: Finkbeiner, B., Wies, T. (eds.) Verification, Model Checking, and Abstract Interpretation. pp. 319–331. Springer International Publishing, Cham (2022)
33. Roh, M., Daigle, B.J.J., Gillespie, D.T., Petzold, L.R.: State-dependent doubly weighted stochastic simulation algorithm for automatic characterization of stochastic biochemical rare events. In: Journal of Chemical Physics. vol. 135. American Institute of Physics (2011)
34. Roh, M., Gillespie, D.T., Petzold, L.R.: State-dependent biasing method for importance sampling in the weighted stochastic simulation algorithm. In: Journal of Chemical Physics. vol. 133. American Institute of Physics (2010)
35. Roh, M.K., Daigle, B.J.: Sparse++: improved event-based stochastic parameter search. BMC Systems Biology **10**(1), 109 (2016). https://doi.org/10.1186/s12918-016-0367-z, https://doi.org/10.1186/s12918-016-0367-z
36. Rosenbluth, M.N., Rosenbluth, A.W.: Monte carlo calculation of the average extension of molecular chains. The Journal of Chemical Physics **23**(2), 356–359 (1955). https://doi.org/10.1063/1.1741967, https://doi.org/10.1063/1.1741967
37. Samoilov, M., Plyasunov, S., Arkin, A.P.: Stochastic amplification and signaling in enzymatic futile cycles through noise-induced bistability with oscillations. Proceedings of the National Academy of Sciences **102**(7), 2310–2315 (2005). https://doi.org/10.1073/pnas.0406841102, https://www.pnas.org/doi/abs/10.1073/pnas.0406841102
38. Soloveichik, D., Seelig, G., Winfree, E.: Dna as a universal substrate for chemical kinetics. Proceedings of the National Academy of Sciences **107**(12), 5393–5398 (2010). https://doi.org/10.1073/pnas.0909380107, https://www.pnas.org/doi/abs/10.1073/pnas.0909380107
39. Villén-Altamirano, J.: Restart vs splitting: A comparative study. Performance Evaluation **121-122**, 38–47 (2018). https://doi.org/https://doi.org/10.1016/j.peva.2018.02.002, https://www.sciencedirect.com/science/article/pii/S0166531616300839
40. Villén-Altamirano, J.: An improved variant of the rare event simulation method restart using prolonged retrials. Operations Research Perspectives **6**, 1–9 (2019). https://doi.org/10.1016/j.orp.2019.100108, http://hdl.handle.net/10419/246387
41. Villén-Altamirano, M., Villén-Altamirano, J.: The Rare Event Simulation Method RESTART: Efficiency Analysis and Guidelines for Its Application, pp. 509–547. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-02742-0_22, http://dx.doi.org/10.1007/978-3-642-02742-0_22
42. Villen-Altamirano, M., Villen-Altamirano, J., et al.: Restart: a method for accelerating rare event simulations. Queueing, Performance and Control in ATM (ITC-13) pp. 71–76 (1991)
43. Wald, A.: Sequential tests of statistical hypotheses. The Annals of Mathematical Statistics **16**(2), 117–186 (1945), http://www.jstor.org/stable/2235829
44. Zhang, B.W., Jasnow, D., Zuckerman, D.M.: Efficient and verified simulation of a path ensemble for conformational change in a united-residue model of calmodulin. Proceedings of the National Academy of Sciences **104**(46), 18043–18048 (2007). https://doi.org/10.1073/pnas.0706349104, https://www.pnas.org/doi/abs/10.1073/pnas.0706349104
45. Zuckerman, D.M., Chong, L.T.: Weighted ensemble simulation: Review of methodology, applications, and software. Annu Rev Biophys **46**, 43–57 (May 2017). https://doi.org/10.1146/annurev-biophys-070816-033834